Breif Projet Git/Github

Introduction:

Dans le cadre de la validation des compétences de la période SAS ; le brief projet est un moyen utile pour que vous validez les compétences dans leur niveau respectif (niveau 1, 2 et 3).

La gestion des workflows sous GIT/GITHUB sera la base de notre apprentissage en termes de la gestion de projet agile

Un scénario doit être respecté en ce qui concerne le déroulement des tâches sous un Product BACKLOG (explication ci-dessous).

Objectifs

- La formation d'un groupe de 5 apprenants
- Planification des tâches selon une méthode de gestion de projet (SCRUM).
- La finalisation des Scénario dans les délais.
- La rédaction d'un rapport d'activités du groupe.
- La manipulation des commandes GIT.
- La découverte du monde GITHUB
- La gestion du temps, du stress et la finalisation des livrables.
- Les réunions de fin de Scénario
- La formalization

Public cibles et contexte

- Les apprenants de la formation YOUCODE
- Les apprenants forment par eux même des groupes de 5 personne

Contexte

Le brief projet sera divisé sur 3 jours :

- La familiarisation avec les commandes GIT.
- La gestion du workflow GIT.
- La gestion du workflow GIT/GITHUB.

Livrables :

1. Un rapport d'activités générales des taches effectués.
2. Une planification journalière des Scénarios qui sont sous forme d'un Product BACKLOG.
3. Un livrable dans le respect des délais.

Les Prérequis

- GITBASH.
- UN compte GITHUB.
- UN tableau Trello.
- Editeur du texte (VSCODE, ATOM, Brackets …).
- P4merge (logiciel pour la comparaison des fichiers).

Lien utiles

- http://www.initializr.com/
- https://stackoverflow.com/
- https://www.perforce.com/products/helix-core-apps/merge-diff-tool-p4merge
- https://gist.github.com/dgoguerra/8258007
- https://gist.github.com/andresmoschini/a127d003885f7e137ef2889fb9e54844

Scénario #1

Préface

- Formez des groupes de 5 apprenants.
- Créez un tableau Trello ou un tableau sous Excel pour la planification du Product BACKLOG et le délai de la livraison de du premier scénario.

- Le Product BACKLOG est déjà priorisé, il suffit juste de maintenir les taches avec le groupe et le délai de la livraison des tâches.
- Un rapport d'activités journalier est exigé.
- Temporalité : toute la journée jusqu'à Minuit.
- Utilisez le git Bash et les commandes pour le premier scénario.

Product BACKLOG

#1: First Step : Immersion

1. Définissez git en quelques mots ?
2. Expliquez le fonctionnement du fichier caché « .git ».
3. Sur votre bureau créez un répertoire nommée « /projects ».
4. Créez un Repository Local sous le nom « /demo ».
5. Créez un Repository Local sous le nom « /demo ».

6. Expliquez le commentaire.
7. Créez le fichier README.md et ajouter la ligne suivante : '#Demo project un simple fichier'.
8. Fait es le staging et le committing avec un commentaire.

#2 : Second Step : La découverte

1. Déplacez-vous dans les fichiers de configuration « .git ».
2. Tapez ls –al
3. Expliquer les clauses suivantes : HEAD, LOGS, BRANCHES
4. Retournez vers votre Repo et créez le fichier « Licence.md ».
5. Faites-le Commit
6. Affichez les fichiers traqués.

#3 : third step : Historique

1. Affichez le dernier commit sur une ligne en ajoutant l'option d'affichage de la hiérarchie de la branche, avec les commit et leur branche aussi
2. Créez un alias de la commande précédente le nom de l'alias est : historique
3. Affichez la liste des alias
4. Affichez l'historique des commit du fichier README.md avec l'alias.

#4 : Fourth Step : Excluding files

1. Renommer le fichier Licence.md à Licence.txt
2. Faites le staging avec mise à jour (ne pas faire « git add . »)
3. Faites-le commit.
4. Supposons que nous développons sur notre plateforme, surement on a des fichiers qu'on veut exclure de notre arborescence du repository Local

a. Créez un fichier nommé application.log
b. Ne faites pas le staging mais créez un fichier nommé « .gitignore »
c. Sur le fichier .gitignore Ajoutez la ligne suivante « *.log »
d. Faites le staging et le commit
e. Qu'est-ce que vous constatez ?

#5 : fifth Step : Branching and Merging

un petit rappel, on a trois types pour faire le merge (The fast forward merging, The manual, The automatic)

1. Modifiez le fichier README.md
2. Ne faites pas le commit
3. Créez une branche pour la modification du fichier README.md du nom 'updates'Faites le staging et le commit en une seule ligne.
4. Affichez l'historique avec l'alias
5. Qu'est-ce que vous constatez ?
6. Retournez vers la branche Master.
7. Affichez l'historique avec l'alias
8. Qu'est-ce que vous constatez
9. Maintenant faites le merge.

#6: sixth Step : Conflict Resolution

1. Créez une branche avec le nom ‘BAD’
2. Modifiez le fichier README.md et ajouter la ligne ‘Trouble’
3. Faites le staging et le commit en une seule ligne
4. Switchez vers la branche principale ‘master’
5. Maintenant modifiez le fichier README.md et ajoutez la ligne ‘Troubleshooting’
6. Staging/committing avec commentaire ‘branche bien faite’
7. Oups !! corriger le commentaire du dernier commit
8. Affichez l’historique
9. Qu’est-ce que vous constatez ? expliquez l’objectif de ce petit scénario ?
10. Faites le merge de la branche ‘BAD’
11. Expliquez le message ?
12. Exécutez la commande suivante : cat README.md ? Expliquez ?
13. Tapez la commande « git mergetool » ? qu’est-ce que vous constatez ?

#7: seventh Step: merge tools

Dans cette section vous êtes amené à installer le logiciel ‘p4merge’. Lien d’installation : https://www.perforce.com/products/helix-core-apps/merge-diff-tool-p4merge

1. Installez p4merge sur votre PC
2. Sur git bash tapez la commande : git config –global –list
3. Git config –global merge.tool p4merge
4. Git config –global mergetool.p4merge.path “lien d’installation” (.exe)
5. Tapez la commande git mergetool
6. Analysez ce que vous voyez ? et expliquez la plateforme ouverte en temps réel ?

#8: eighth Step: Challenge

Après avoir résolu les conflits des branches ; vous avez maintenant des fichiers indésirables à rejeter :

- Sur le fichier .gitignore ; écrivez une clause pour rejeter les fichiers indésirables et redondants
- On laisse que les fichiers : licence.txt Readme.md .gitignore (exemple *.log pour application.log)

N’oubliez pas de faire vos Commits

Mots clé : Git ls-files ; git config –global –list ; git log –oneline –graph –decorate –all ; git checkout –b “nom branche”

Scénario #2

Préface

- Vous avez terminé la phase #1 du brief
- On va voir des nouveaux concepts avancés
- Restez sur la même disposition des groupes

- Le Product BACKLOG est déjà priorisé, il suffit juste de maintenir les taches avec le groupe et le délai de la livraison des tâches
- Un rapport d'activités journalier est exigé
- Temporalité : toute la journée jusqu'à Minuit.

Product BACKLOG #2

#1: First Step : Tagging

1. Déplacez-vous sur la branche Principale
2. Créez un TAG avec un nom V1.0 et un commentaire ' RELEASE 1.0 '
3. Affichez les informations sur le TAG
4. D'après vous un TAG il sert à faire quoi au juste ?

#2: Second Step: Stashing and saving work in Progress

1. Modifiez le fichier README.md , ajoutez une ligne
2. A vrai dire, l'équipe a décidé que cette tâche sera en standby et que les modifications progressent avec le timeline du projet

a. Tapez la commande git Stash
b. Expliquez le fonctionnement de la commande git Stash
c. Tapez la commande git stash list
d. Exécutez la commande git status ?
e. Qu'est-ce que vous constatez ?

3. Modifiez le fichier « Licence.txt », ajoutez la ligne « APACHE 2.0 »
4. Faites staging et le commit en une seule ligne
5. Exécutez la commande « git stash pop » ?
6. Qu'est-ce que vous constatez ?

#3: third step: Voyage sur Github, Local Repo to github Repo

1. Pour cette phase vous devez avoir un compte GITHUB
2. Créez un repo github public sans ajouter le fichier README.md
3. Créez le remote en https
4. Examinez le remote
5. Pushez le tout à travers la commande : git push –u origin master - -tags

a. Expliquez la commande ?

b. Expliquez les options –u et - -tags ?

6. Sur Github Vérifiez la liste des commits , les branches , les releases et les tags

#4: Fourth Step: Mini challenge (optionnel)

En examinant les types d'authentification sur GITHUB, on tombe sur l'authentification HTTPS et SSH, certes HTTPS est beaucoup plus facile à manager tandis que le SSH est plus sécurisé tandis que fiable en ce qui concerne les transferts cryptés.
Le but de ce challenge est de créer une authentification SSH entre votre repo local et le repo GITHUB.

- HINTS :
  - Créez un dossier sous git nommé .SSH
  - Déplacez-vous dans le fichier .SSH
  - Maintenant à vos mains : Créez une authentification sécurisé SSH entre votre repo local et votre repo distant (aide : Ssh –keygen-t rsa –C ' email').

#5: Fifth Step: Création d'une local copy

1. Sur github créez un autre repo nommé (Monsiteweb)
2. Ajoutez à l'arborescence toujours sur github le fichier .gitignore et un fichier alicence.txt 'APACHE 2.0 '
3. Déplacez-vous dans le répertoire projets sous GIT
4. Créez un clone Github vers le local sous le nom (Monsiteweb-local)
5. Vérifiez si le clone est créé.

#6: sixth step : Sending the website :

1. Télécharger le site web depuis le lien suivant : http://www.initializr.com/
2. Sur le site telecharger un site bootstrap avec le fichier .htaccess et le fichier 404.html
3. Analysez l'arborescence.
4. Expliquez ?
5. Copiez le site télécharger dans votre repo local à travers une seule commande :
6. Hint : cp –R '~/DESKTOP/initialzr/*' . (Supposons que vous êtes sur votre repo local)
7. Git status
8. Faites le staging et le commit en une seule ligne
9. Faite le push à github
10. Vérifiez l'existence du site local sur votre repo github

#7: seventh step: Fetch and pull :

1. Sur github éditez le fichier Index.html, sur la balise <title> </title> ajoutez le titre, mon premier site web.
2. Faites le commit sur github
3. Sur git et sur le repo local, Editez le fichier README.md
4. Faites le staging et le commit en une seule ligne
5. Qu'est-ce que vous constatez ?
6. Trouble shooting :

a. Executez la commande « git fetch »
b. Git status
c. Expliquez?
d. Git pull
e. Git push
f. Verifiez les commit sur github

L'objectif de cette phase : c'est lorsqu'il y'a deux changements sur le repo local et sur le repo github, donc les démarches décrites c'est pour la gestion de ce conflit

Fin du deuxième scenario, Maintenant vous avez préparé l'environnement GITHUB pour des concepts plus avancés, ainsi que vous avez acquis comment dialoguer avec GITHUB en se basant sur les commandes GIT

Mots clés

Ssh –keygen-t rsa –C ' email' ;

Scénario #3

Préface :

- ⇨ Vous avez terminé la phase #2 du brief
- ⇨ Vous êtes maintenant capable de travailler avec GIT et GITHUB en même temps
- ⇨ Vous êtes capable de créer la première arborescence professionnelle (README.md , Licence.txt , .gitignore) et vous êtes capable de configurer le fichier .gitignore .
- ⇨ Restez sur la même disposition des groupes
- ⇨ Maintenant let's build !!

Le Product BACKLOG est déjà priorisé, il suffit juste de maintenir les taches avec le groupe et le délai de la livraison des taches

Un rapport d'activités journalier est exigé

Temporalité : toute la journée jusqu'à Minuit.

Utilisez le gitBash et les commandes pour le premier Scenario

Product BACKLOG:

#1: First Step: Changes on Github

- ⇨ Sur votre compte Github : faites les modifications suivantes :
  1. Sur le dossier CSS ajoutez un autre fichier nommée style.css
  2. Sur index.html ajoutez juste après la balise <body> une balise <H1> : <h1> modification récente </h1>
  3. Renommer le fichier 404.html en error404.html.

- ⇨ Maintenant la synchronisation avec votre repository local, les changements effectués doivent impérativement apparaitre sur votre repo LOCAL :
  1. Sur gitHub vérifiez la liste des commits .
  2. Sur GITBASH, vérifiez avec « git status »
  3. Visualisez et télécharger les fichiers distants sur GITBASH
  4. Maintenant faites le pull et fusionnez les changements distants avec le repo local
  5. Vérifiez votre repo Local
  6. Sur votre repo Distant GITHUB, laissez un commentaire sur les fichiers que vous avez modifiez en regardant la liste des commits.
  7. Sur GITBASH affichez les informations du commit de l'ajout de la balise <h1></h1>

#2: Second Step: Branching and merging sur GITHUB

1. Repo distant :
   - ⇨ Sur GitHUB créez une branche du nom 'Example'
   - ⇨ Sur la branche 'Example', modifiez le fichier READM.md et faites-le commit (toujours sur GITHUB)
   - ⇨ Sur github et sur le menu 'branch' ; Expliquez ce que vous voyez comme message ?

2. Repo local :
   - ⇨ Sur votre repo local, créez une branche nommé 'annulation'.
   - ⇨ Sur votre repo local éditez le fichier « index.html » et supprimez la balise H1.
   - ⇨ Faites-le commit et le stagging en une seule ligne.
   - ⇨ « Git show » pour vérifier vos changements.

3. Vérification :
   - ⇨ Sur GITBASH Pushez les changements que vous avez sur la branche 'annulation' à votre repo distant
   - ⇨ Maintenant vérifiez votre repo distant.
   - ⇨ Qu'est-ce que vous constatez.

#3: Third Step: compare pull Requests

- ⇨ Sur votre repo Distant, un message indiquant qu'une branche a été ajoutez avec un bouton « compare and pull request »
- ⇨ Cliquez sur le bouton ? analysez le rendu de la page ? qu'est-ce que vous constatez ?
- ⇨ Créez un pull request avec le commentaire 'annulation à vérifier'
- ⇨ Une page s'ouvre, explorez la page ? qu'est-ce que vous constatez ?
- ⇨ Sur l'onglet 'conversation' vous pouvez aussi laisser des commentaires à la personne qui demande le pull request ; laissez un commentaire ?

- ⇨ Maintenant si tout va bien ; aucun message d'erreur effectuez le merging ?
- ⇨ Supprimez la branche 'annulation' une fois tout est OK.

#4: fourth Step: merging en local

Pour finaliser l'objectif de cette étape, on va créer une branche nommé 'ajustement', ou on va modifier le fichier 'main.css' (css/main.css) en ajoutant la ligne suivante :

```
body {

   Width : 100% ;
   Height : 100%
}
```

- ⇨ Faites maintenant le nécessaire pour 'pushez' les modifications sur GITHUB.
- ⇨ Sur votre GITHUB, un pull request est demandé !
- ⇨ Maintenant, essayez de faire le merge en local ! @ vos mains !
    1. Sur GITBASH, Vérifiez dans quelle branche vous êtes maintenant ! surement sur la branche 'ajustement'
    2. Un pull Request est demandé sur GITHUB ! N'oubliez pas !
    3. Sur GITBASH, revenez vers la branche master ! Expliquez le rendu de la commande ?
    4. Maintenant demandez le pull ?
    5. Une fois la commande du pull est exécutée, à votre avis quel type de merge on aura besoin pour faire un merge sans conflits
    6. Faites le merge à travers GITBASH
    7. Une fenêtre s'ouvre après l'exécution de la commande du merge ; ce qui signifie le message délivré ? à quoi sert cette fenêtre ?
    8. Git status
    9. Si tout est OK ! Pushez les changements ?
    10. Si tout est OK ! sur GITHUB le pull sera vérifier ? explorez le pull pour visualiser les changements ?

- ⇨ Sur GITHUB supprimez les branches que vous avez créé.
- ⇨ Sur le local on doit apporter les changements qu'on a fait sur GITHUB :
    1. Tapez la commande git branch –a ? expliquez le message ?
    2. Sur GITBASH supprimez les branches que vous avez créé.

3. Maintenant ! Tapez la commande git bash –a ? qu'est-ce que vous constatez ?
4. Ce que nous devons faire maintenant, est de faire appelez la suppression faites sur GITHUB et l'appliquez sur le local ! Trouvez la commande et exécutez ?
5. Tapez la commande git bash –a ? qu'est-ce que vous constatez ?

#5: fifth Step: The Cleaning up

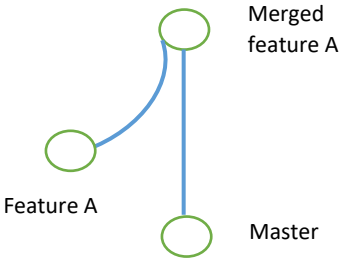
Dans cette phase je vous laisse découvrir l'objectif ?

- ⇨ Sur GITHUB Créez une branche nommée 'updatelicence, éditez le fichier Licence et changer Apache 2.0 par Apache 3.0
  - ⇨ Sur GITBASH faites un pull global : git pull –-all ;
  - ⇨ Maintenant faites le merge de la branche 'updatelicence'.
  - ⇨ Executez Git push !
  - ⇨ Git branch –d 'updatelicence'
  - ⇨ Exécutez la commande git bash –a ? qu'est-ce que vous constatez sur le remote et aussi sur GTHUB?

- ⇨ Maintenant sur GITBASH, trouvez la commande qui va nous permettre de supprimer la branche depuis le remote et ainsi de pushez les changements de la suppression simultanément ? Vérifiez vos Repo (local et distant)
- ⇨ Pourriez-vous expliquez l'objectif de cette phase ?

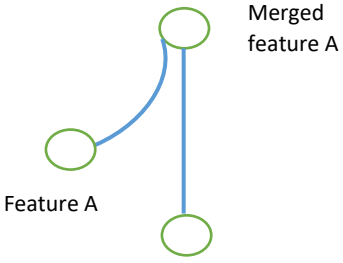
#6: sixth step: Rebasing:

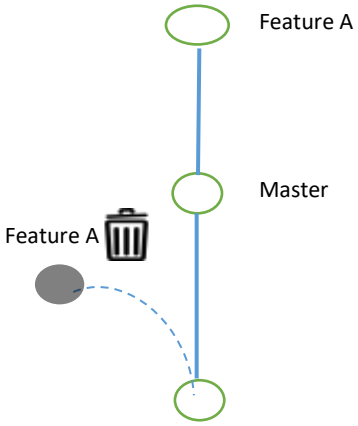
⇨ Introduction générale : Rebase vs merge



Quand faire un merge ?

- ⇨ Sur des branches durables
- ⇨ Pour indiquer un évènement marquant : Release, Sprint etc.





Quand faire rebase ?

- ⇨ Sur acceptation d'un pull Request d'un feature ou suivi d'un merge
- ⇨ Lors d'un git pull

- ⇨ Faites une veille sur REBASE de 30 min intergroupe , visitez le lien suivant : https://www.grafikart.fr/tutoriels/amend-rebase-588
- ⇨ Sur GITHUB, Editez le fichier README.md, ajoutez la ligne 'updates' ! n'oubliez pas le commit
- ⇨ Sur GITBASH Editez le fichier index.html supprimez les lignes suivantes :

```
<!--[if lt IE 7]>      <html class="no-js lt-ie9 lt-ie8 lt-
ie7" lang=""> <![endif]-->
<!--[if IE 7]>         <html class="no-js lt-ie9 lt-
ie8" lang=""> <![endif]-->
<!--[if IE 8]>         <html class="no-js lt-ie9" lang=""> <![endif]-->
<!--[if gt IE 8]><!-->
```

Et modifiez le titre du site : choisissez le titre que vous vouliez

- ⇨ Faites le stagging et le commit en une seule ligne
- ⇨ Maintenant apportez les changements faites sur le Repo distant : fetch
- ⇨ Git status ? lisez les commentaires
- ⇨ Maintenant si vous compreniez le commentaire, on a besoin de faire un rebase : git pull - - rebase
- ⇨ Exécutez l'alias que vous avez créé dans le scénario #1 (historique) ? qu'est-ce que vous constatez sur l'arborescence des branches ?
- ⇨ Vérifiez que tous les changements que vous avez faits sont OK ?
- ⇨ Expliquez l'utilité du REBASE ?

#7Seventh step : GitHub Insights:

Sur Github vous avez un menu INSIGHTS ; Explorez votre activité à travers insights :

- ⇨ Analysez le Network ou le timeline de votre workflow ?
- ⇨ Discutez entre vous équipe et formalisez ce que vous voyez sur le Network en une production Timeline sur Tableau à marqueur ou trello : imaginez aussi que chaque membre du groupe à contribuer dans une tache du Timeline Network (Like a Sprint planning) .

#8: eigtht step: Default branch and conflicts

- ⇨ Sur github créez une branche du nom "Dev"
- ⇨ La Branche Dev deviendra la branche par défaut ; sur le menu « settings » mettez la branche Dev en mode par défaut
- ⇨ Créez une autre branche « demo », on va la considérer comme « feature ».
- ⇨ Avant de faire un pull Request sur le branche demo , Editez le fichier « README.md » , Ajoutez une ligne
- ⇨ Revenez à la page d'accueil de votre repo ? à votre avis on a besoin dans ce cas de faire un merge ou bien un rebase ? pourquoi ?
- ⇨ Ne faites aucun merge ni Rebase mais plutôt supprimez la branche feature depuis GITHUB
- ⇨ Sur GIT, supprimez récursivement et localement votre repo website
- ⇨ Maintenant Clonez via https ou SSH si vous l'avez réussi
- ⇨ Vérifiez si le contenu est récupéré

- ⇨ Vérifiez les branches et ou pointe le pointeur : HEAD
- ⇨ Maintenant on a besoin de la branche master comme base : git checkout master
- ⇨ Toutefois la branche par défaut est : ……

Quick Challenge: Try and Catch ()

On va imaginer le mini Scénario suivant :

- ⇨ Sur GitHub, on va éditez le fichier Readme.md
- ⇨ Localement on est sur la branche DEV.
- ⇨ On a fait des changements sur le REPO distant, surement on a besoin d'un fetch

- ⇨ On va provoquer un sous-entendu, on va éditez le fichier README.md localement en sachant qu'on a déjà modifié le fichier sur le repo distant.
- ⇨ On va faire le commit
- ⇨ Normalement, un des membres de l'équipe demande un pull : git pull
- ⇨ Un message s'affiche indiquant que GIT n'arrive pas à traquer les informations et qu'il y'a un problème du merging, vu que le merge automatique n'est pas autorisé

@ votre avis comment vous allez résoudre le problème évoqué.

- HINTS : mergetool , 3 way merge

Mot clés :

Git fetch, git push –u origin master, git pull, git branch –a; git branch –d ; git push origin :option ;